

Designing Distributed User Interfaces for Ambient Intelligent Environments using Models and Simulations

Kris Luyten^{a,*}, Jan Van den Bergh^a, Chris Vandervelpen^a,
Karin Coninx^a

^a*Hasselt University – transnationale Universiteit Limburg,
Expertise Centre for Digital Media,
Wetenschapspark 2, 3590 Diepenbeek (Belgium)*

Abstract

There is a growing demand for design support to create interactive systems that are deployed in ambient intelligent environments. Unlike traditional interactive systems, the wide diversity of situations these type of user interfaces need to work in require tool-support that is close to the environment of the end-user on the one hand and provide a smooth integration with the application logic on the other hand. This paper shows how the Model-Based User Interface Development methodology can be applied for ambient intelligent environments; we propose a task-centered approach towards the design of interactive systems by means of appropriate visualizations and simulations of different models. Besides the use of these typical user interface models such as the task- and presentation-model to support interface design, we focus on user interfaces supporting situated task distributions and a visualization of context influences on deployed, possibly distributed, user interfaces. To enable this we introduce an environment model describing the device configuration at particular moment in time. To support the user interface designer while creating these complex interfaces for ambient intelligent environments, we discuss tool support using a visualization of the environment together with simulations of the user interface configurations. We also show how the concepts presented in the paper can be integrated within Model-Driven Engineering, hereby narrowing the gap between HCI design and software engineering.

Key words: Distributed user interfaces, UML, model-driven engineering, task migration

* Corresponding author. Address: Hasselt University, Expertise Centre for Digital Media, Wetenschapspark 2, 3590 Diepenbeek (Belgium)
Email addresses: kris.luyten@uhasselt.be (Kris Luyten),

1 Introduction

¹ Modern middleware solutions allow mobile and embedded software components to communicate with each other while residing on heterogeneous platforms. Modern middleware also offers automatic discovery mechanisms to locate necessary software and hardware available in an ubiquitous environment. While this can be considered as a step toward the ubiquitous computing vision Mark Weiser predicted Weiser (1991), there still exists a large gap between the actual tasks a user should be able to perform and the user interfaces exposed by ubiquitous systems to support those tasks. This gap is caused mainly by different missing pieces that are necessary to have a generic approach towards the creation of ubiquitous user interfaces. We address some of these missing pieces in this paper: we will employ a model-based and a model-driven approach to enable a smooth integration between the creation of the user interface and the development of the application logic.

Distributable user interfaces enable the user to exploit more and new possibilities of an ambient computing environment by allocating tasks to interaction resources that best support those tasks. We define an *interaction resource* as an atomic I/O channel that enables communication between user and system. Atomic indicates the I/O channel is supported in one way only (e.g. from user to system or from system to user) and is limited to a single modality. Examples of interaction resources are keyboards, mice, all sorts of screens, speech synthesizers, force feedback devices, . . . Usually, an interaction resource is advertised in an environment through the computing device it is attached to. This computing device is called an *interaction cluster* and manages input from or output to interaction resources attached to it. The aforementioned definitions imply also a multi-modal user interface is composed of different interaction resources, not necessarily located on the same interaction cluster.

During the last couple of years many research papers have been published discussing requirements, frameworks and models for distributed user interfaces, e.g. Balme et al. (2004); Larsson and Berglund (2004); Savidis et al. (2002); Vandervelpen and Coninx (2004), but there is still a lack of tools to allow designers to create such interfaces. The design of a user interface that can be distributed over several interaction resources in an ubiquitous computing environment is a tedious task and has not yet been addressed extensively. The creation for decent tools to support the design of ambient intelligent user interfaces is essential: design tools can hide the technical details and high complexity of the target environment for designers. For example, distributed

jan.vandenbergh@uhasselt.be (Jan Van den Bergh),
chris.vandervelpen@uhasselt.be (Chris Vandervelpen),
karin.coninx@uhasselt.be (Karin Coninx).

¹ Original article available at <http://dx.doi.org/10.1016/j.cag.2006.07.004>.

interfaces are typical for supporting interaction in ambient intelligent environments but require detailed knowledge of networking and distributed systems, something we want to hide from the designer.

In this paper we present our work on a task-centered methodology for the *design* and the *deployment* of ambient intelligent user interfaces. For this purpose we have created *MoDIE* (Mobile Distributable Interface Engineering), a tool that relies on Human-Computer Interaction (HCI) models introduced by Model-Based User Interface Design (MBUID). In addition, by integrating support for UML 2.0 based models in MoDIE, a rigorous software engineering process can be established. As such our approach provides the opportunities of UML-based modeling methodologies and tools whilst bridging the gap between traditional software engineering models and the models from model-based user interface development. Such an integration can be a first step towards integration of model-based design within model driven engineering approaches.

MBUID has already been used extensively to develop multi-device and even context-aware user interfaces Eisenstein et al. (2001); Clerckx et al. (2004b); Mori et al. (2004). In MBUID, different abstract models such as the task model and the domain model highlight different aspects of the user interface independent of details of the target devices. Concrete models such as the presentation model and navigation model will contribute more specific details towards the presentation of the interface. Typically, the complexity of these models is proportional with the complexity of the target domain. For ambient intelligent environments, models tend to be very complex and thus require a suitable translation into intuitive interactive tool support for the designer to work with.

The remainder of this paper is structured as follows: section 2 gives an overview of the related work that defines the underlying concepts for the topic of this paper. Next, section 3 discusses the different aspects that need to be taken into account to support a task-centered approach to design user interfaces for ambient intelligent environments. Section 4 explains how context can have a big influence on the task execution and what needs to be done to anticipate this while modelling. Section 5 presents the design tool we are developing to support the design process, followed by a discussion of the opportunities that are available when integrating UML-based modeling. Finally, section 7 gives a conclusion.

2 Related Work

ICrafter Ponnekanti et al. (2001) is a framework of services and their user interfaces for use in an ubiquitous environment. Services can register their

user interfaces with the ICrafter Interface Manager. This way, other service can request those interfaces through the Interface Manager for rendering purposes. Once the interface is instantiated, interaction with the associated service becomes possible. This service-oriented approach provides a uniform and location-independent access to the functionality of the system. Dynamic composition or on-the-fly aggregations of user interface components are central to this approach. However, there is no design support to constrain the dynamic behavior so it is difficult to ensure the user interface is usable while supporting the envisioned tasks.

Heider and Kirste Heider and Kirste (2002) propose a goal-driven approach to decide which interaction resources to use. In their approach a planning algorithm is used for developing strategies to reach the predefined goals. An execution control component can execute a strategy and manages the resources that are necessary for the selected strategy. This approach is useful to cope with the enormous complexity of designing a user interface that should work in an ambient intelligent environment. A task-centered approach could benefit by using a planning algorithm to calculate an optimal strategy for executing the required tasks with the interaction resources that are available. Look et al proposed a similar approach in Look et al. (2003), where the importance of the user's goals is recognized as input for deciding on an optimal resource allocation to support the user.

Distribution of a user interface among different interaction resources or multiple surfaces is also gaining importance: unlike traditional desktop computing, a user interface in an ambient intelligent environment is no longer limited to one device that is the center of interaction. In Coutaz et al. (2003) an ontology for multisurface interaction is proposed by Coutaz et al. This ontology offers an unifying framework for reasoning about distributed user interfaces. Because of the complexity of the covered types of problems, this ontology can only show its full potential when it is used in a HCI design tool.

Balme et al. Balme et al. (2004) presented the CAMELEON-RT Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. Some type of middleware is provided (the Distribution-Migration-Plasticity middleware) to allow smooth integration of user interfaces that reside on different physical locations. This type of support for distributed user interfaces is required to deploy a user interface for an ambient intelligent environment. In Vandervelpen et al. (2005) we show how conventional interactive websites can be distributed among different interaction resources with a minimum of effort required from the designer to prepare the website for distribution. This proves a structured high-level user interface description language (HTML in the case of the website) is the most suitable way to create distributable user interfaces.





Modeling user interfaces is an important part of the design of complex interactive applications. The user interface, however, must also be coupled to application logic. This together with the fact that often programmers must ultimately also produce working code for the user interface, led to several approaches that describe models that originally came from the MBUID community using UML.

Some of the earliest results in pairing UML diagrams and model-based user interface design were discussed in Nunes (2000). UMLi and Wisdom were among the discussed approaches. UMLi da Silva and Paton (2003) focused on the description of user interfaces and defined two new diagram types for the description of user interfaces. The presentation model was described using a diagram similar to the deployment diagram, while the user interface logic was described using a notation based on the activity diagram. The Wisdom approach Nunes and e Cunha (2000), a light-weight software-engineering method for small businesses, used a set of stereotypes to extend the UML for interactive systems development. Among others it used an extended version of the class diagram to express the presentation model and the task model. CanonSketch Campos and Nunes (2005) offers specialized tool support for the presentation model offering views using UML, Canonical Abstract Prototype notation Constantine (2003) and HTML.

The Context-sensitive User interface Profile (CUP) Van den Bergh and Coninx (2005a,b) is a UML 2.0 profile targeting the development of context-sensitive user interfaces, combining features of both Wisdom and UMLi while taking advantage of the extra capabilities that UML 2.0 over the earlier versions used by both Wisdom and UMLi.

3 Properties of Ambient Task Modelling

3.1 Task Notation and Dialog Derivation

We use Paternò's ConcurTaskTrees (CTT) notation Paternò (2000); Mori et al. (2002); a notation for task modeling that provides temporal operators between tasks. This notation offers a graphical syntax, an hierarchical structure and a notation to specify the temporal relation between tasks. Four types of tasks are supported in the CTT notation: abstract tasks , interaction tasks , user tasks  and application tasks . These tasks can be specified to be executed in several iterations. Sibling tasks, appearing in the same level in the hierarchy of decomposition, can be connected by temporal operators like choice (\square), independent concurrency ($|||$), concurrency with information exchange ($||\square||$), disabling ($[>$), enabling ($>>$), enabling

with information exchange ($\llbracket \gg \rrbracket$), suspend/resume ($\llbracket \gg \rrbracket$) and order independence ($\llbracket = \rrbracket$). Paternò and Santoro (2002) specifies the following priority order among the temporal operators: *choice* $>$ *parallel composition* $>$ *disabling* $>$ *enabling*. Figure 1 shows a simple example of a CTT specification for querying information about a person.

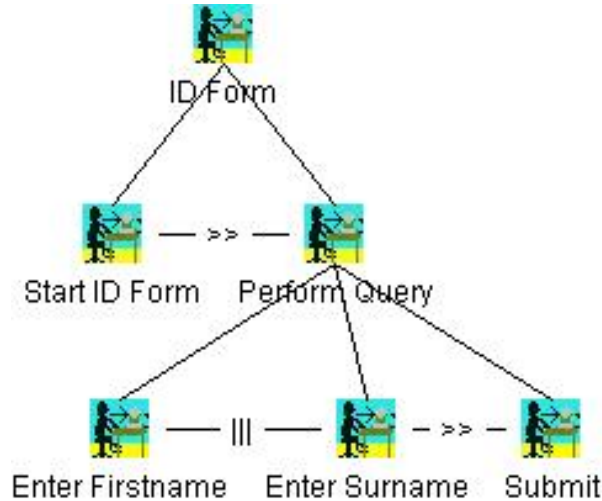


Fig. 1. Example CTT specification

The CTT notation allows to extract task sets where each task set contains tasks that should be “active” during the same period of time in order to reach a (sub)goal. This concept is called *enabled Task Sets* (TSs) Paternò (2000). For a given task model M several of such enabled task sets can be identified: each set contains tasks that execute within the time frame defined by the set and do not overlap with other tasks from other sets. We can describe this process by the function $f : M \rightarrow TS_1, TS_2, \dots, TS_n$ that maps a task model M on the set of enabled task sets $TS_{i,1 \leq i \leq n}$; which is a set of subsets of the model M . Several design tools exist that provide this TS extraction functionality by means of the ConcurTaskTrees notation and their use is described in existing literature Mori et al. (2002, 2004); Luyten et al. (2003); Vanderdonckt et al. (2003).

Each task set $TS_{i,1 \leq i \leq n}$ contains a subset of tasks t_1, t_2, \dots, t_m from the task model M . A task set requires a distribution configuration for the tasks it contains: the representation of a task set is distributed among different devices that are available in the environment. Notice that a user interface distribution is defined in section 1. It specifies the combination of tasks in a dialog with the available interaction devices. Temporal relations between different tasks, together with the fact there are no two TSs that can overlap in time, allow to construct a sequence of TSs that the user(s) should execute to reach their goals as specified by the task specification. Figure 2 depicts an example of such a sequence of enabled task sets (labeled with TS1, TS2, ...).

3.2 Task-set Distributions

The first property we consider in our approach is *completeness*. User interface completeness indicates that all interaction tasks needed to reach a goal at a particular moment are made accessible to the user regardless of the interaction resources available in the environment (including the interaction resources exposed by the user’s personal devices). The use of TSs to guide the design process ensures this property: all tasks of the active TS need to be allocated to interaction resources that can handle these tasks. From a given task model the number of TSs that can be found is exactly the minimal number of logically different interfaces (or “presentation units” according to Eisenstein et al. (2001)) the designer should provide to allow the user to access the complete functionality of a system. Figure 2 shows how tasks in an active TS are distributed over interaction resources in the environment. Notice TSs can be ordered in time because of the definition given above (this ordering is also referred to as the dialog model).

The second property we consider is *continuity*. User interface continuity ensures the user can interpret and evaluate the internal state of the system while using different interaction resources. Consequently, the user does not lose track of the current task. When the distribution of the interface parts changes at run time, this property must hold. Consistency of the user interface across different platforms can support a better continuity of the user interface. In this sense continuity is a broader concept: it refers to minimization of interruption of the user by the changes in the user interface. Providing support for the preservation of continuous interaction will pose a difficult challenge for a design methodology (and tool) that uses tasks, activities and temporal relations P. Faconti and Massink (2000). In our approach continuity is supported by constraining the possible task-distribution strategies. For example; a constraint to support continuity is the *fixed task constraint* which is formalized as follows: if the tasks in TS_i are enabled and $\exists t \in M : t \in TS_i \wedge t \in TS_j$ then t will not be re-distributed to another device when a transition from TS_i to TS_j is executed. A task that reoccurs in different TSs can be restricted to the same device when the transition to the following TS is made. In figure 2, *task 3* is an example of the application of such a fixed task constraint for the transition from TS_5 to TS_3 .

We can add more specific constraints depending on the properties of the devices. For $t \in M$, t can be constrained to a set of devices D_c that is a subset of all available devices D , and $\forall d \in D_c, \pi_c(d) = 1$. $\pi_c(d)$ is a projection of the property c over the element d . E.g.: when distributed, certain tasks can be constrained to devices that have some kind of network communication available. In this example the property value is 0 if there is no network communication available and 1 otherwise. Of course, a distribution can also be constrained

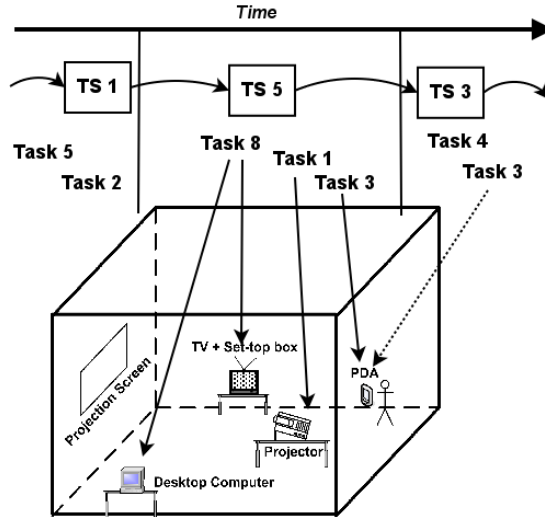


Fig. 2. Different Enabled Task Sets on a timeline with their distribution

according to a value of a property such as the quality of network communication that is available. This can be expressed as $b_1 \leq \pi_b(d) \leq b_2$, where b is the attribute of d representing the bandwidth available at element d , b_1 is the lower boundary of the required bandwidth and b_2 the upper boundary.

The properties such as the bandwidth should be made explicit in the design tool. This allows the designer to use these properties while modelling the interactive system. Section 5 shows how the device model and constraints are combined in an environment model and used in a tool to support task-modelling for ambient intelligent environments. The environment model can also be represented as a UML deployment diagram that encodes the available interaction resources, relations between interaction resources and properties of both resources and relations. Section 6 shows the relation of the environment model and the deployment diagram.

3.3 Task Migration Paths

The previous section discussed the distribution of tasks of each individual TS taking into account continuity and completeness of the user interface. Once an appropriate set of task-set distributions is found for each TS, the designer should be able to constrain the transitions from one TS to another. A lack of continuity because of a context switch (other devices that come into play, tasks that appear and disappear,...) can have a disastrous effect on task performance.

In traditional MBUID this is represented by a dialog model and the transitions between different dialogs. These transitions could be invoked by simple interactions such as a window manipulation Vanderdonckt et al. (2003). In

an ambient intelligent environment things are more complicated however: the *physical location of the user interface parts* differs from one dialog to another in contrast with a single-device system where a dialog is always represented on the same device. The design of such a system should make sure the cognitive burden of making a transition is minimized while supporting the tasks and goals of the user. Denis and Karsenty Denis and Karsenty (2004) describe a set of design principles to ensure inter-usability in a multi-device environment: inter-device consistency, transparency and adaptability of device usage. In this paper we focus on the first principle to support task set transition continuity. Inter-device consistency is composed out of four levels: perceptual (appearance and structure), lexical (labeling), syntactical (operations) and semantic (service functionality) consistency. The former two levels, perceptual and lexical consistency, are provided by the presentation model that is used. The latter two levels, syntactical and semantic consistency, can be enforced by defining a set of constraints in the environment model as shown in the previous section. The support of these types of consistency levels inside the different models contributes to a better continuity while making the transition from one TS to another.

3.4 Task Representations

Each interaction task from the task specification should be presented in the environment one way or another so the user can interact with it. In particular the interaction tasks can be annotated by different ways they can be presented to the user(s). For each task $t \in M$ an abstract user interface description $x \in \{X_1, X_2, \dots, X_n\}$ can be retrieved, the set of related user interface descriptions is referred to as the *presentation model*. Based on the findings in related research Luyten et al. (2004), a user interface description is specified using an XML-based notation. Figure 2 shows how the high-level user interface descriptions of *all* tasks available in an TS are distributed among different appropriate interaction resources available in the environment while the user continues her/his interaction with the application, from one active TS to the other.

For each TS there are different possibilities of how the user interface representing the set of tasks can be divided. In Vandervelpen et al. (2005) we showed a method that uses XHTML as the presentation language and a set of rules and a cost function to select the “preferred” distribution configuration among all possible configurations. The XHTML document was subdivided according the tasks it supported, and the different parts were distributed among the available devices in the neighborhood of the user.

4 Contextual Task Constraints

The allocation of a task to a set of interaction resources can also constrain the execution of the task. For example: a task can only be valid within a certain physical range because the interaction resource it is allocated to, has to maintain a communication channel with another device that executes a parallel task exchanging information with the first task. Figure 3 shows this scenario. In Clerckx et al. (2004a) we presented an approach to take these kind of context switches explicitly into account in the task and the dialog model. A decision task can be inserted in the task model: this type of task allows a designer to specify a set of rules that can select an alternative task set to execute according to the context of use. This approach allows us to insert a decision rule in the task specification that will select another task set when the device is out of range.

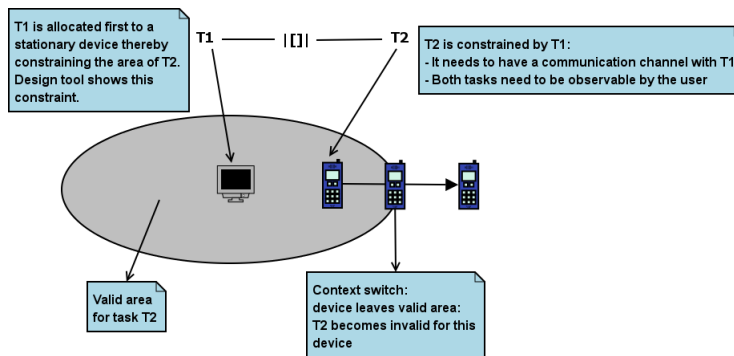


Fig. 3. Location constraint example for the task specification $T_1 | [] | T_2$.

To support this kind of reasoning for a task-centered approach we need to extend the semantics of the task specification with new constraints besides the temporal constraints and hierarchical structure. More precisely: we need to relate the context of use and the task set in terms of constraints over the task distribution behavior. Elaborating on the example of figure 3, where there are two tasks that can be executed in parallel and exchange information while performing $(t_1 | [] | t_2)$, two different constraints can be identified for these two tasks:

- (1) both tasks should be observable at the same time by the user
- (2) both tasks should be able to exchange data using some kind of communication channel.

The first one depends on the designer's intentions and should be part of the task specification, the second one can be derived from the task-device allocations. With either one (or both) of these constraints there is only one possibility: the device that represents t_2 has to be located in the predefined area of the device presenting t_1 . Notice t_1 and t_2 belong to the same TS, since they can be

executed during the same period of time. This example is equally valid for the construction $t_1 | \square | t_2 | \square | \dots | \square | t_n$ (t_1, \dots, t_n belong to the same task set), but the number of constraint checks involved to evaluate a distribution configuration for all tasks increases to $\binom{n}{2}$ in this situation. If the number of areas increases to m , the number of constraint checks increases exponentially since there are now $\binom{n}{m}$ possible combinations. The number of possibilities that a designer would have to check by hand is not feasible without any tool support. Our approach allows to visualize these constraints and automatically define valid task distribution configurations according to the task specification.

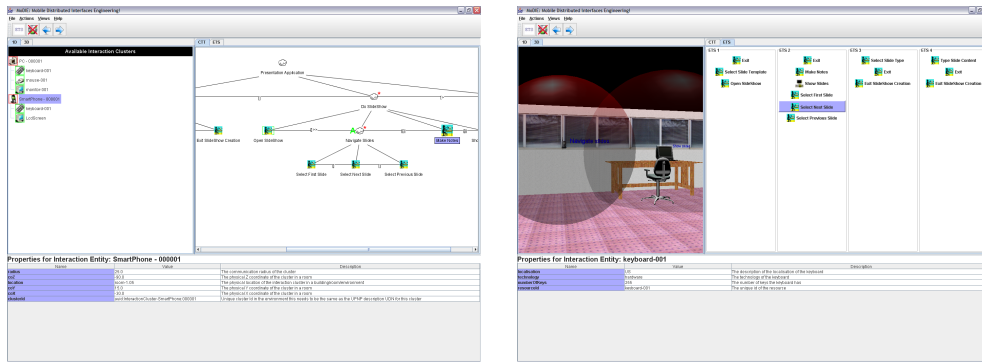
The example in the previous paragraph focused on a typical intra-TS relation: a relation between two tasks in the same TS. It is sufficient to take this into account for a distribution configuration for a single TS. In other cases however, similar concerns arise. For example, the construction $t_1 \square \gg t_2 \square \gg t_3 \square \gg \dots \square \gg t_n$ implies that every task is in a separate TS, but still requires each task $t_{i, 1 \leq i \leq n-1}$ to exchange information with its successive task $t_{i+1, 1 \leq i \leq n-1}$. These types of relations have to be taken into account for the possible migration paths between task sets.

5 The MoDIE Platform

The models discussed in the previous section are all integrated by the MoDIE platform, a platform that supports a user interface design process for ambient intelligent environments. The central model is the *task model*, describing the set of tasks the (ubiquitous) application supports. Other models include the environment model that describes all available interaction resources in the environment of the user, a dialog model containing the TSs derived from the task model, a presentation model that is related to the tasks in the task model and an interaction model describing the interaction between the user interface and the application logic. Every view in MoDIE offers direct manipulation of these different models and visualizes the relations between different models appropriately. Figure 4(c) shows an environment view combined with a task view that allows to assign tasks to interaction resources.

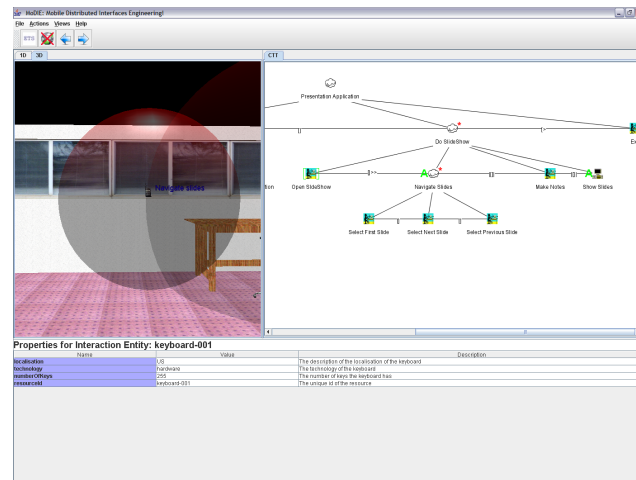
Figure 5 shows the MoDIE platform architecture. It consists of three modules: The MoDIE design tool (MDT), an Environment Repository (ER) and the Distribution Manager (DM). When the ER is started, it will probe the ambient intelligent environment to look for interaction clusters that can be used in an interactive distributed user interface session (*figure 5(1)*). At the moment, this is implemented using Universal Plug and Play (UPnP²). We extended the UPnP discovery mechanism such that the interaction clusters can respond

² <http://www.upnp.org>



(a) View on the Interaction Resources of the environment model

(b) Visualizing a distribution configuration for a task set



(c) Allocating tasks from a task specification to devices in an ambient intelligent environment

Fig. 4. Different views of the MoDIE tool.

by sending a description of their properties. This description is provided in the form of a Resource Description Format (RDF, <http://www.w3.org/RDF/>) document that is being sent to the ER. By using RDF, the ER can use existing tools (like Jena³) to parse all the received documents and merge them into one in-memory RDF model. From now on, this in-memory RDF model will be referred to as the Environment model. Notice that this model is based on the CoDAMoS ontology presented in Preuveneers et al. (2004). Figure 6 shows the RDF model of an interaction cluster. For keeping the figure concise, the properties for the interaction resources are omitted. For example, the *codamos:keyboard001* interaction resource has properties *localization*, *keys* and *technology*.

Once the ER is started, the MDT can interface with it (*figure 5(2)*) locally or through a network to get information about the environment model. The MDT

³ <http://jena.sourceforge.net/>

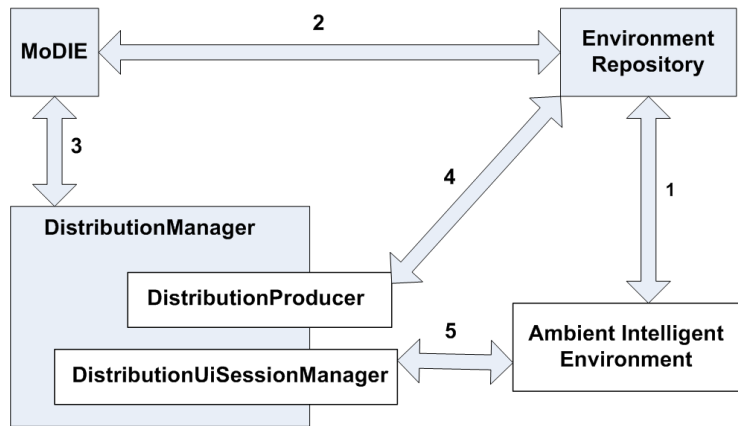


Fig. 5. The MoDIE platform architecture

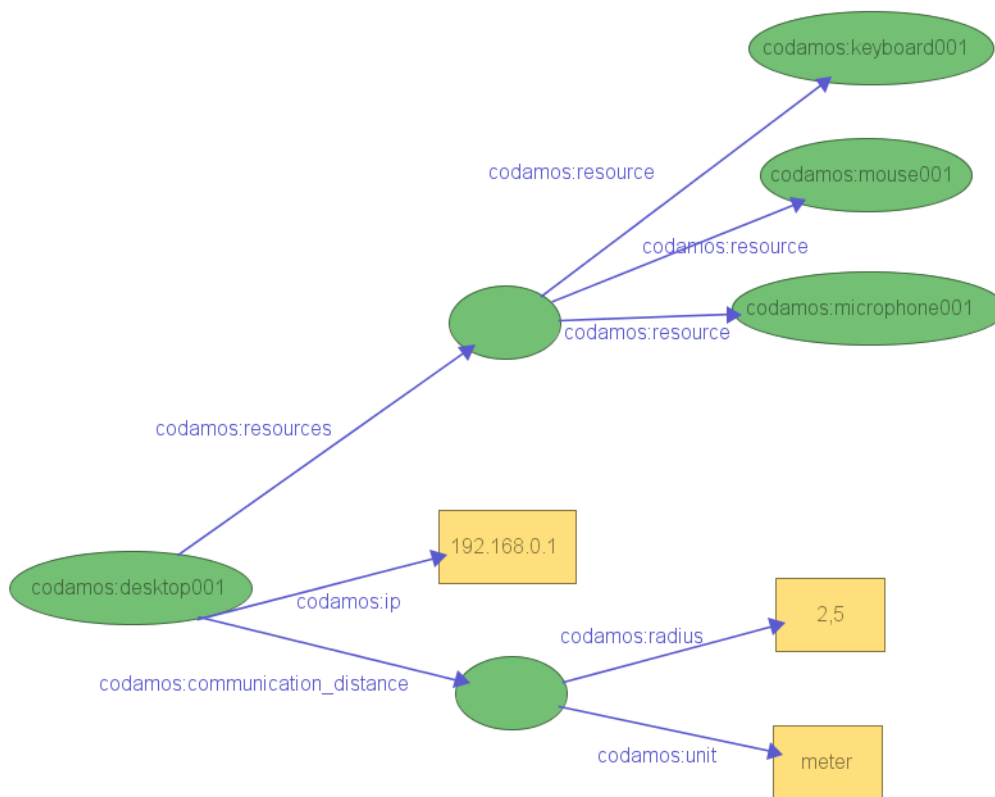


Fig. 6. A piece of the environment RDF graph

uses this information to build a visual representation of the environment and to check constraints of interaction resources while the UI designer constructs the distribution model. A constraint check on a property of an interaction resource can be translated into an RDF query that is executed on this RDF model. Part of the RDF query is just an implementation of the projection function of section 3.2 which maps an interaction resource property on a value in the domain of this property. Another part of the RDF query reflects the

condition on the value of this property. Notice that the UI designer also has the ability to create an environment model through the MDT. This is interesting because it opens the doors to simulate environments for testing purposes.

The last important module of the MoDIE platform is the DM which can be used to deploy the user interface in the real ambient intelligent environment according to the distribution model constructed by the UI designer (using the MDT) and the state of the environment. The distribution model describes the properties of interaction resources that can be used for particular tasks in the task model. The DM can be invoked by the MDT (*figure 5(3)*). In this mode, the necessary models are passed to the DM and the state of the task distributions at a particular moment is communicated back to the MDT. The MDT uses this information to update its visualizations. This helps the UI designer in understanding the distributed user interface evolution as reactions to changes in the environment. However, the DM can also be used on its own as part of a runtime infrastructure for deploying distributable user interfaces that were designed using the MDT. This is done by providing the serialized models as input to the DM.

The DM is also divided into two modules. The first one is the DistributionProducer (DP). The DP is also connected to the ER and uses RDF queries to check constraints upon the properties of the interaction resource in the environment model (*figure 5(4)*). These constraints are described in the distribution model. Using this technique, a possible distribution scenario is calculated and passed to the DistributedUiSessionManager (DSM). The DSM is responsible for executing the scenario by sending the appropriate user interface components to the interaction clusters as depicted in the current distribution scenario (*figure 5(5)*). Notice that the DP can register itself to receive notifications from the ER when changes in the properties of relevant interaction resources occur (like for example the position). These notification can result in a new distribution scenario that is passed to the DSM.

Tasks can be related with interaction resources of the environment model in two ways:

- (1) Automatically: task can be allocated among the available interaction resources automatically by applying the different constraints.
- (2) Manually: usually, there are a number of solutions that are valid with respect to the constraints defined by the different models. The MDT supports manual editing of the task allocations (which actually presents the task-environment inter-relation): the designer can relate tasks with interaction resources and observe the effects of these changes.

An important aspect of the MDT is the possibility to simulate the run-time behavior of the distributed user interface. This simulation is considered as a

view on the different models that are built with the MDT and is integrated with the other views. The simulation creates a 3D model of the environment model (using the Java3D API⁴), and uses the list of interaction resources to dynamically render the user environment. A simulation module aids in defining the appropriate Task Migration Paths. Figure 4(a) shows the MoDIE combined view of the environment model and the dialog model (expressed as a set of task sets). By moving the mobile device away from the desktop computer the designer can see what kind of transitions are invoked and how the design fits in the simulated situation.

Relating tasks with devices through direct manipulation on the 3D view of the environment model is obviously more intuitive than working only with diagrammatic notations. Although this model supports direct manipulation, it is also suitable to visualize existing relations already created between the other models. This way the designer will have a graphical overview of the user interface distribution and instantly sees the effect of model manipulations.

A possible extension that is investigated is to use a UML representation of the ER. This could be done in real-time within the tool or by converting snapshots of the ER to a XMI-document that can be imported and displayed in many of the currently available UML-modeling tools. Conversely a UML-diagram, describing a certain task distribution could serve as input for a simulation by converting it to RDF.

6 Integration with UML-based Software Engineering

This section details a mapping of the previously introduced concepts to UML 2.0 Object Management Group (2004) and how this mapping can be used to combine the approach with model-driven engineering. We made the choice to use UML 2.0 in order to have a rigorous description of the different aspects of interactive software based on an established technology. The fact that it has better facilities for model-driven development which can bring a boost to design methodologies based on a diverse set of models is promising.

An integration with the UML modeling languages is important for several reasons: first of all it makes aids in bridging the gap between the HCI designer and the software developer. In particular for the complex domain described in this paper (ambient intelligent environments) there is currently no support to integrate the design of application logic with the interaction design. Besides the fact UML is widely accepted, it also offers a more formal way to describe the functionality of a system and it provides the tools to relate a user interface

⁴ <http://java.sun.com/products/java-media/3D/>

with the functionality that is represented by this user interface.

6.1 Mapping to UML

To represent the architectural aspects of a distributed user interface, we propose the use of UML 2.0 deployment diagrams. The deployment diagrams can be used to describe some features in more detail, which cannot be seen easily in the 3D view. One part of the architecture is the communication channels that are available between the different interaction clusters. Another is the composition of an interaction cluster; which interaction resources are contained in the cluster. Traditionally, the deployment diagram is a static diagram, but in the current setting this diagram depends on the context-of-use. Section 5 introduced the dynamic discovery of available interaction clusters: the result of such a discovery can be visualized as a deployment diagram. Since the content of the deployment diagram depends on the point in time when a discovery is executed, it is possible to have many deployment diagrams for a single application.

The distribution of a user interface can be specified by allocating parts of the user interface to specific interaction resources or interaction clusters. There is a natural mapping from interaction clusters and interaction resources to Devices in UML 2.0, where the former can contain other Devices (interaction resources) and the latter cannot, due to the definition of an interaction resource (see section 1). A part of a user interface can be allocated to a certain device by specifying the manifest-relationship between the part (a stereotyped class) and an Artifact on a specific node.

Stereotypes can be used to denote the function of each user interface part (`<<inputComponent>>`, `<<outputComponent>>`, or `<<actionComponent>>`). Parts can be combined in a `<<groupComponent>>`. An `<<inputComponent>>` part is used to allow the user to give input but also contains labels, drawings or sound that is necessary or aids in the understanding of what information should be put into the system. Selection from a non-empty set of options is also considered input. An output part is a part of the user interface that shows information provided by the application core, including all relevant labels etc. An `<<actionComponent>>` is a part of the user interface that is responsible for triggering functionality in the application core. Note that `<<inputComponent>>` and `<<groupComponent>>` are not mutually exclusive, allowing selection of a group of user interface parts related to a single concept. In a graphical user interface such a selection could correspond to the selection of a table-row. More information about these stereotypes and the related models they are used in can be found in Van den Bergh and Coninx (2005b).

We thus propose to explicitly model the links between the logical structure of the user interface and the physical structure. Specific stereotypes, such as `<<html>>` or `<<javaSwing>>` can be applied to the Artifacts to identify the technologies that will be used to realize them. This approach has the advantage that the realization of the user interface is explicitly and unambiguously specified using an approach already in use for the application logic.

In figure 7 the scene and allocations from figure 2 are represented using the UML deployment diagram. In this diagram, interaction clusters and interaction resources are represented by Nodes, while the physical representation of the task, the user interface through which the task can be performed, is depicted as an Artifact linked to a class with an appropriate icon indicating the type of task. An integration of this approach into MoDIE allows the designer to get adapted representations of the deployment diagram for the different possible situations. The use of stereotypes and the associated tagged values can allow adapted representations in standard UML modeling tools, should MoDIE support saving configurations manually or semi-automatically selected by the designer into XMI Group (2002). The advantages and disadvantages of both approaches are currently under investigation.

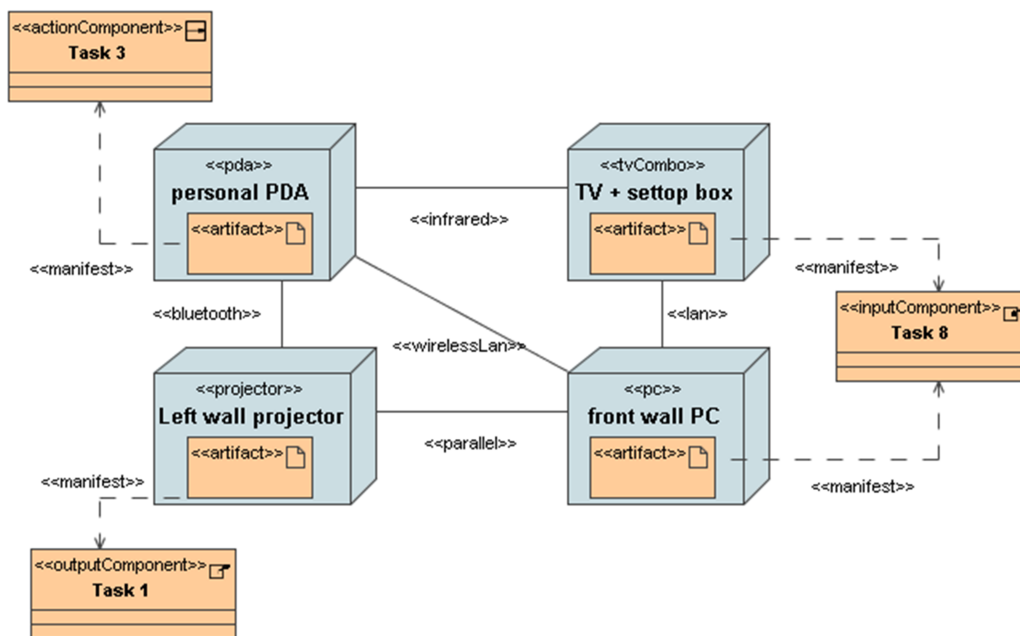


Fig. 7. Deployment diagram describing the task allocation of figure 2

6.2 Model-driven engineering

Abstract presentations can be derived from the tasks/task sets. These can be converted to concrete, albeit high-level description based on marks (stereo-

types) made to the abstract models and context information (e.g. user profiles). The resulting concrete representation can be similar to the notation used for Canonical Abstract Prototypes Constantine (2003), possibly including an allocation to interaction resources.

The use of abstractions can be useful to derive user interfaces that are consistent and complete, but have different appearances. It would be more difficult to design a multi-cultural, multi-device user interface that is both consistent and complete using separate designs. The use of model-driven development starting from a high-level (platform independent) model that is refined through one, or possible multiple, transformations into a concrete user interface to drive or guide the design of a user interface can offer many benefits.

An example can be the realization of a wizard-based user interface on a kiosk system versus a single form on a wall-sized screen operated by his smart phone when both user interfaces could be used to perform the same tasks. At the highest level, both interfaces are represented using the same set of user interface components. Using multiple transformations, this high-level model is gradually translated into concrete models. This can be done by applying HCI design patterns or best practices based upon contextual information.

We envision two possible tool configurations for integration of MoDIE with model driven engineering to create a complete environment for the design of distributed user interfaces. In both configurations MoDIE provides the task-based distribution facilities based upon the discovery of interaction devices and resources in the neighborhood and designer input. MoDIE works at the task-level and relies on the linking of tasks to (declarative) user interface descriptions (using URI's) to accomplish effective distribution of user interfaces.

In the first configuration all necessary components to create the user interfaces are integrated into one integrated tool, as can be seen in figure 8(a). In this configuration, the distribution created in the MoDIE tool is passed to a separate part of the tool that works with UML and starts from a deployment specification as discussed in the previous section. Starting from this model, several transformations will be made that gradually transform the abstract model into different concrete models, specifying concrete user interface configurations for certain hardware configurations. These concrete models can then be translated into XML-based user interface descriptions of which the URI's can be delivered back into the MoDIE tool to create an actual deployment. The main advantage of this approach is that there is only one environment to develop and that it can be easier to get specific support for transformations and some modeling needs. It however implies that a significant amount of work already established for other environments has to be duplicated.

The second configuration splits the total functionality over three parts (see

figure 8(b)): MoDIE (1) for task distribution, once this task distribution is created, it is delivered in XMI-format to an UML-tool (2) supporting model driven engineering (MDE). After one or more transformations, the abstract representation that was given to the UML-tool is translated into one or more concrete models (of user interfaces) and is delivered to a user interface modeling tool that provides the look-and-feel to the user interfaces through the use of constraints and styles. The advantage of this second configuration is that one can use existing tools for software engineering tools for manipulating the models, integrated development environments such as MagicDraw, and transformation tools. In this configuration it might however be more difficult to tackle problems that are specific for the design of distributed user interfaces. An example is offering synchronized alternative views of one or more models, such as models that describe the user interface structure.

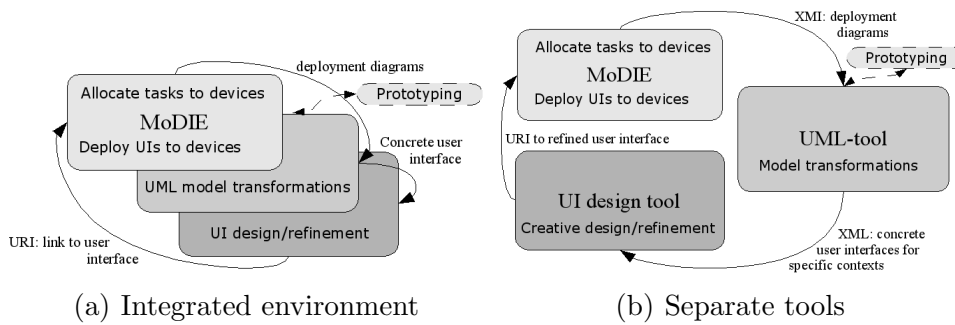


Fig. 8. MoDIE tool and model driven engineering.

7 Conclusions and Future Work

In this paper we investigated the requirements to support model-based user interface design for ambient intelligent environments: a promising approach that abstracts user interface design away from the technical details typical for an ambient intelligent environment. It enables us to design and test user interfaces that can be distributed among different interaction resources. For this purpose we introduced MoDIE, a system that uses a model-based approach to design user interfaces for ambient intelligent environments. MoDIE allows a designer to combine different models, such as the presentation model, the task model and an environment model, that describe different aspects that influence the final user interface but release the designer from the technical details necessary to realize the distributed user interface. This tool visualizes task allocations in a 3D representation of the real environment and makes the design process more intuitive by using visualization and simulation techniques. User interface completeness (is the required functionality to reach the user's goals accessible?) and continuity (can we create a usable user interface for a dynamic environment?) are the two main properties that are considered

here. Both the visualization of the task allocations in the environment and the simulation of the execution of a task specification are the primary tools to ensure completeness and continuity.

It is clear there are an overwhelming number of aspects that need to be taken into account to use a model-based approach for designing user interfaces that are deployed in ambient intelligent environments. Traditional model-based user interface development approaches do not take dynamic environments with different devices that can be used simultaneously into account. This work contributes to a solution for this problem by proposing new ways to aid the designer in creating user interfaces for complex environments and hiding the technical details so one can focus on the tasks the user interface should support instead of the implementation issues involved..

Acknowledgements

The authors would like to thank Geert Houben, Frederik Winters and Tim Clerckx for co-developing the software supporting the ideas of this paper.

Part of the research at EDM is funded by EFRO (European Fund for Regional Development), the Flemish Government and the Flemish Interdisciplinary institute for Broadband technology (IBBT). The CoDAMoS (Context-Driven Adaptation of Mobile Services) project (IWT 030320) is directly funded by the IWT (Flemish subsidy organization).

References

- Balme, L., Demeure, A., Barralon, N., Coutaz, J., and Calvary, G. (2004). CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In Markopoulos, P., Eggen, B., Aarts, E. H. L., and Crowley, J. L., editors, *EUSAI*, volume 3295 of *Lecture Notes in Computer Science*, pages 291–302. Springer.
- Campos, P. F. and Nunes, N. J. (2005). Canonsketch: a user-centered tool for canonical abstract prototyping. In Rémi Bastide, Philippe Palanque, Jörg Roth, editor, *Engineering Human Computer Interaction and Interactive Systems: Joint Working Conferences EHCI-DSVIS 2004*, volume 3425 of *LNCS*, pages 146–163. Springer.
- Clerckx, T., Luyten, K., and Coninx, K. (2004a). DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In *The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems*.

- Clerckx, T., Luyten, K., and Coninx, K. (2004b). Generating Context-Sensitive Multiple Device Interfaces from Design. In *Pre-Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces CADUI'2004, 13-16 January 2004, Funchal, Isle of Madeira, Portugal*.
- Constantine, L. L. (2003). Canonical abstract prototypes for abstract visual and interaction design. In *Proceedings of DSV-IS 2003*, number 2844 in LNCS, pages 1 – 15, Funchal, Madeira Island, Portugal. Springer.
- Coutaz, J., Lachenal, C., and Dupuy-Chessa, S. (2003). Ontology for multi-surface interaction. In *INTERACT*.
- da Silva, P. P. and Paton, N. W. (2003). User interface modelling in umli. *IEEE Software*, 20(4):62–69.
- Denis, C. and Karsenty, L. (2004). *Inter-Usability of Multi-Device Systems – A Conceptual Framework*, pages 373–385. Wiley.
- Eisenstein, J., Vanderdonck, J., and Puerta, A. R. (2001). Applying model-based techniques to the development of uis for mobile computers. In *Intelligent User Interfaces*, pages 69–76.
- Group, O. M. (2002). Omg xml metadata interchange. Object Management Group, WWW, <http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf>.
- Heider, T. and Kirste, T. (2002). Supporting Goal-Based Interaction with Dynamic Intelligent Environments. In van Harmelen, F., editor, *ECAI*, pages 596–600. IOS Press.
- Larsson, A. and Berglund, E. (2004). Programming ubiquitous software applications: requirements for distributed user interface. In Maurer, F. and Ruhe, G., editors, *SEKE*, pages 246–251.
- Look, G., Peters, S., and Shrobe, H. (2003). Plan-Driven Ubiquitous Computing. In *Artificial Intelligence in Mobile System*.
- Luyten, K., Abrams, M., Limbourg, Q., and Vanderdonck, J., editors (2004). *Developing User Interfaces with XML: Advances on User Interface Description Languages*.
- Luyten, K., Clerckx, T., Coninx, K., and Vanderdonck, J. (2003). Derivation of a Dialog Model for a Task Model by Activity Chain Extraction. In Jorge, J. A., Nunes, N. J., and F. e Cunha, J., editors, *DSV-IS*, volume 2844 of *Lecture Notes in Computer Science*, pages 203–217. Springer.
- Mori, G., Paternò, F., and Santoro, C. (2002). CTTE: support for developing and analyzing task models for interactive system design. *IEEE Trans. Softw. Eng.*, 28(8):797–813.
- Mori, G., Paternò, F., and Santoro, C. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520.
- Nunes, N. J., editor (2000). *Towards A UML Profile for Interactive Systems Development (TUPIS 2000)*. online.math.uma.pt/tupis00/.
- Nunes, N. J. and e Cunha, J. F. (2000). Towards a uml profile for interaction design: the wisdom approach. In Evans, A., Kent, S., and Selic, B., editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard.*, vol-

- ume 1939 of *LNCS*, pages 101–116. Springer.
- Object Management Group (2004). *UML 2.0 Superstructure Specification*.
- P. Faconti, G. and Massink, M. (2000). Continuity in Human Computer Interaction. In *CHI 2000 Workshop report*. <http://www.acm.org/sigchi/bulletin/2000.4>.
- Paternò, F. (2000). *Model-Based Design and Evaluation of Interactive Applications*. Springer.
- Paternò, F. and Santoro, C. (2002). One model, many interfaces. In Kolski, C. and Vanderdonckt, J., editors, *Computer-Aided Design of User Interfaces III CADUI*, volume 3, pages 143–154. Kluwer Academic.
- Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., and Winograd, T. (2001). ICrafter: A Service Framework for Ubiquitous Computing Environments. In *UbiComp 2001: Ubiquitous Computing, Third International Conference Atlanta, Georgia, USA, September 30 - October 2, 2001, Proceedings*, Lecture Notes in Computer Science, pages 56–75. Springer.
- Preuveneers, D., den Bergh, J. V., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., and Bosschere, K. D. (2004). Towards an Extensible Context Ontology for Ambient Intelligence. In Markopoulos, P., Eggen, B., Aarts, E. H. L., and Crowley, J. L., editors, *Ambient Intelligence: Second European Symposium, EUSAI 2004, Eindhoven, The Netherlands, November 8-11, 2004. Proceedings*, pages 148–159.
- Savidis, A., Maou, N., Pachoulakis, I., and Stephanidis, C. (2002). Continuity of interaction in nomadic interfaces through migration and dynamic utilization of I/O resources. *Universal Access in the Information Society*, 4(1):274–287.
- Van den Bergh, J. and Coninx, K. (2005a). Towards Modeling Context-Sensitive Interactive Applications: the Context-Sensitive User Interface Profile (CUP). In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 87–94, New York, NY, USA. ACM Press.
- Van den Bergh, J. and Coninx, K. (2005b). Using uml 2.0 and profiles for modelling context-sensitive user interfaces. In Pleuss, A., Van den Bergh, J., Hussmann, H., and Sauer, S., editors, *Proceedings of Model Driven Design of Advanced User Interfaces 2005*, volume 159 of *CEUR Workshop Proceedings*, Montego Bay, Jamaica. online CEUR-WS.org/Vol-159/paper7.pdf.
- Vanderdonckt, J., Limbourg, Q., and Florins, M. (2003). Deriving the Navigational Structure of a User Interface. In Rauterberg, M. and Wesson, J., editors, *Proceedings of 9th IFIP Conf. on Human-Computer Interaction Interact'2003 (Zrich, 1-5 September 2003)*, pages 455–462.
- Vandervelpen, C. and Coninx, K. (2004). Towards model-based design support for distributed user interfaces. In *Proceedings of the third Nordic Conference on Human-Computer Interaction*, pages 61–70. ACM Press.
- Vandervelpen, C., Vanderhulst, G., Luyten, K., and Coninx, K. (2005). Light-weight Distributed Web Interfaces: Preparing the Web for Heterogeneous Environments. In *5th International Conference on Web Engineering (ICWE'2005)*. <http://research.edm.luc.ac.be/cvandervelpen/>

research/icwe2005/
Weiser, M. (1991). The Computer for the 21st Century. In *Scientific American*.